Flower Image Classifier – Ayush Ashutosh Panigrahi

CWID – 10469348

Problem Statement & motivation :

Sometimes you see a flower and you guess it as one of the well-known common types like lily, rose, sunflower etc but even flowers have a hierarchy like the animal kingdom where there are different species and subspecies like lily belongs to the family of Liliaceae which in itself consists of 15 different genera. For example, lily and tulips belong to the same species which makes it difficult at times to distinguish them especially when they have the same color. Therefore, I decided to leverage the accurate image classification ability of the deep learning model like CNN's to classify different species of flowers.

About Dataset :

I am using a dataset built by the oxford university which contains images of 102 species of flowers. The flowers are chosen to be blooms found in the United Kingdom. Each class has between 40 and 258 photos. The photographs vary in scale, position, and lighting. Furthermore, there are categories with significant variances within the category as well as multiple extremely similar categories. The data is split and saved as such in different folders respectively with their label mapping saved in a JSON file.

Link: https://www.robots.ox.ac.uk/~vgg/data/flowers/102/

Steps taken in this project :

1. Loading the data :
   Here I have used torchvision to load the data. The dataset has three sections: training, validation, and testing. The data for each set (train, validation, test) is loaded with torchvision's ImageFolder. The data for each set is given to the model with torchvision's DataLoader which is a data generator. I used transformations like random scaling, cropping, and flipping for the training. This will assist the network in generalizing, resulting in improved performance. As needed by the pre-trained networks, the input data is scaled to 224x224 pixels. The validation and testing sets are used to assess the model's performance on new data. We don't want any scaling or rotating modifications for this, but we do need to resize and crop the photographs to the right size. The pre-trained networks I'll be utilizing were trained on the ImageNet dataset, which was normalized independently for each color channel. I have standardized the picture means and standard deviations to what the network expected for all three sets. It's [0.485, 0.456, 0.406] for the means and [0.229, 0.224, 0.225] for the standard deviations, determined using ImageNet pictures. These numbers, which range from -1 to 1, will centre each color channel at 0.

2. Loading the label mapping :
   A mapping from category label to category name is loaded. You can find this in the file cat_to_name.json. It's a JSON object which you can read with the JSON module. This will give you a dictionary mapping of the integer encoded categories to the actual names of the flowers.

3. Model Building :
   A pretrained network such as VGG16 is loaded from torchvision.models and the parameters are frozen. A new feedforward network is defined for use as a classifier using the features from the VGG16 as input, In this case, I have used 2 hidden layers on top of the pre-trained models like VGG16. The first hidden layer has 512 nodes with ReLU activation and a dropout regularization with a 50% probability. The second hidden layer before the output layer has 100 nodes. The output layer consists of 102 layers with the activation logsoftmax as we need to predict the probability of whether the flower belongs to one of the 102 species of flowers.

4. Training the model:
   The parameters for the pre-trained model are frozen as it has been tuned to extract features by training on a large dataset like the ImageNet dataset. Only the parameters of the hidden layers and the output layer built by me on top of the pre-trained model are updated during the training. Feedforward and backpropagation are performed for 10 epochs/iteration with the accuracy achieved on the validation set being printed after every 50 batches (a batch of 8 is used for the weight updates as it would lead to faster convergence). After the training is completed, the trained model is saved as a checkpoint along with associated hyperparameters, the class_to_idx dictionary which makes inference easier later, number of epochs as well as the optimizer state.

5. Testing the model :
   There is a function that successfully loads a checkpoint and rebuilds the model. The process_image function successfully converts a PIL image into an object that can be used as input to a trained model. The predict function successfully takes the path to an image and a checkpoint, then returns the top K most probable classes for that image. A matplotlib figure is created displaying an image and its associated top K most probable classes with actual flower names and their probabilities.

<u>Conclusion and Future Work :</u>

The model achieved a validation accuracy of 83.25 % after 10 epochs which took 7 hours and 35 minutes. I have used a batch size of 8 for faster convergence but this leads to sometimes missing the global minima or getting stuck in a local minimum, hence a learning rate scheduler was used to adaptively decrease the learning rate to omit this issue. I have saved and submitted the .pth file with all the model parameters for the sake of quick inference by just loading it without training the model again. The accuracy can be further increased by using a deeper model like the Densenet121 or a complex model like the resnet or inception model for feature extraction. The model takes a lot of time for training which causes periodic monitoring to be done to check if the model is overfitting, this can be put in check by implementing early stopping on the model which will monitor any metric of our choice and see if it is not increasing/decreasing for a specific period of the epochs and stop the training algorithm to save the resultant model.